Exhibit 3

Exhibit 2

U.S. Patent No. 7,519,814 vs. Oracle

Accused Instrumentalities: Oracle products and services using secure containerized applications, including without limitation Oracle Cloud Infrastructure ("OCI") and Oracle Kubernetes Engine ("OCE"), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

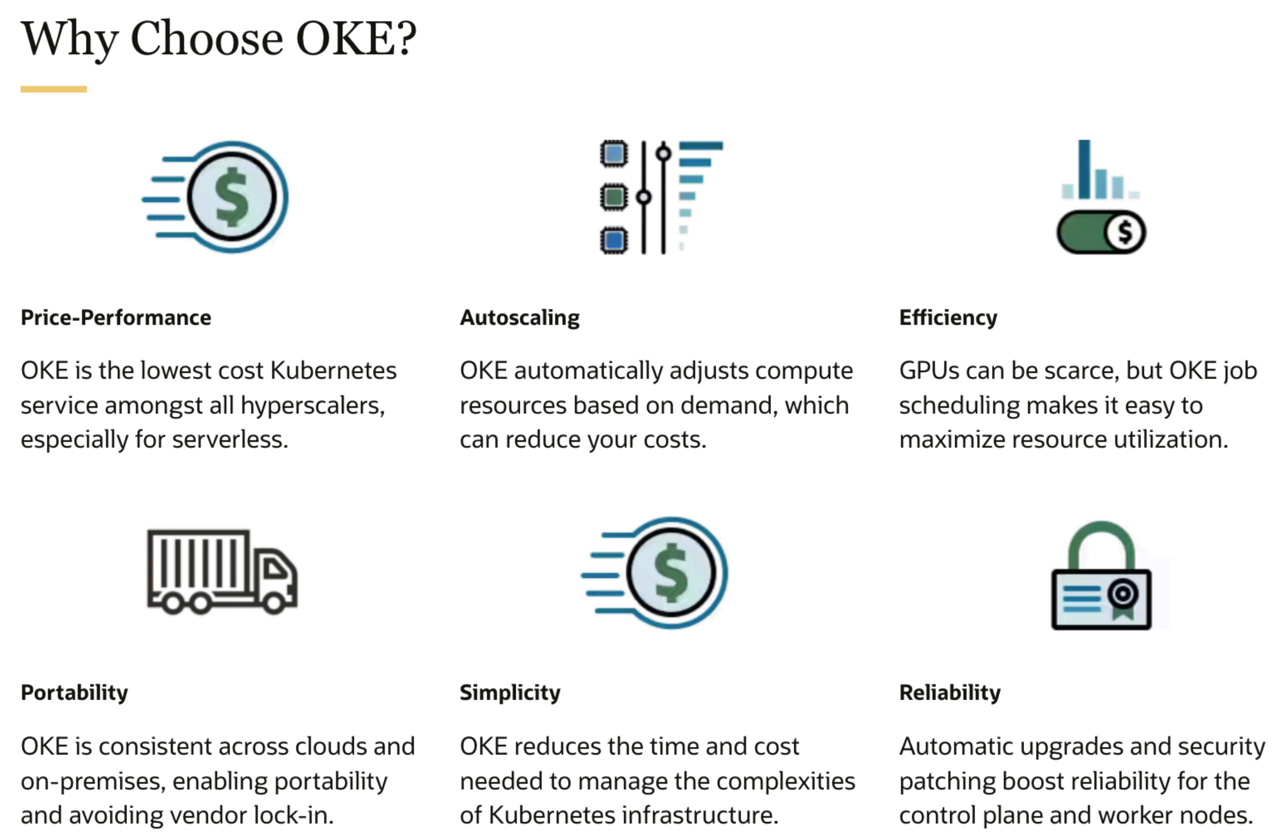
Claim 1	Accused Instrumentalities
[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:	To the extent the preamble is limiting, Oracle and/or its customer practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed. For example, Oracle Kubernetes Engine runs on individual servers, each of which runs an independent operating system running either on bare metal, through an on-premises virtualized infrastructure, through one or more cloud services, or through any other supported deployment. In an exemplary deployment, two or more servers use different operating systems. The servers operate in disparate computing environments, including because each server is a stand-alone computer and/or each server is unrelated to the other servers due to having independent hardware and, in some instances, independent software. Oracle requires and/or provides that each server includes a processor with one or more cores available to the OS kernel. Oracle further requires and/or provides that each server has a supported operating system, which includes a kernel and associated local system files, including for example libraries such as libc/glibc, configuration files, etc. In the infringing system, at least two servers have different operating systems. In at least some instances, Oracle directly owns, operates, controls, and/or benefits from the claimed system and/or method. In other instances, Oracle's customer makes and uses the system and/or method either by following Oracle's direction and control, including Oracle's documentation, or automatically through the ordinary and expected operation of Oracle's software, or a combination thereof.

Case 5:24-cv-04740-PCP Document 23-4 Filed 02/10/25 Page 4 of 56

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>

See claim limitations below.

See also, e.g.:



Why Choose OKE?

Price-Performance

OKE is the lowest cost Kubernetes service amongst all hyperscalers, especially for serverless.

Autoscaling

OKE automatically adjusts compute resources based on demand, which can reduce your costs.

Efficiency

GPUs can be scarce, but OKE job scheduling makes it easy to maximize resource utilization.

Portability

OKE is consistent across clouds and on-premises, enabling portability and avoiding vendor lock-in.

Simplicity

OKE reduces the time and cost needed to manage the complexities of Kubernetes infrastructure.

Reliability

Automatic upgrades and security patching boost reliability for the control plane and worker nodes.

https://www.oracle.com/cloud/cloud-native/kubernetes-engine/

</td></tr>
</table>

Claim 1	Accused Instrumentalities
	Welcome to Oracle Cloud Infrastructure Oracle Cloud Infrastructure (OCI) is a set of complementary cloud services that enable you to build and run a range of applications and services in a highly available hosted environment. OCI provides high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network. https://docs.oracle.com/en-us/iaas/Content/GSG/Concepts/baremetalintro.htm **Existing applications can benefit by migrating to OCI and OKE** OKE offers lower total cost of ownership and improved time to market. OKE simplifies operations at scale in the following ways: – Lift and shift; there's no need to rearchitect – Increase resource utilization and efficiency – Reduce operations burden with automation – Improve agility, flexibility, uptime, and resilience – Save time on infrastructure management – Reduce compliance risk and enhance security https://www.oracle.com/cloud/cloud-native/kubernetes-engine/#app-migration

Claim 1	Accused Instrumentalities
	What is OCI Kubernetes Engine (OKE)? Oracle Cloud Infrastructure Kubernetes Engine (OKE) is a managed Kubernetes service that simplifies the development, deployment, and operation of containerized workloads at scale. OKE enables you to quickly create, manage, and consume Kubernetes clusters that leverage underlying OCI compute, networking, and storage services. **When should I use OKE?** You should use OKE when you want to leverage Kubernetes to deploy and manage your Kubernetes-based container applications. It allows you to combine the production-grade container orchestration of standard upstream Kubernetes with the control, security, and high, predictable performance of OCI. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **How does OKE provide resiliency?** When you create an OKE cluster, OKE automatically creates and manages multiple Kubernetes control plane nodes spread across fault domains and availability domains (logical data centers). This is done to help ensure that the managed Kubernetes control plane is highly available. Control plane operations, such as upgrading to newer versions of Kubernetes, can be performed without service interruptions. Additionally, when you provision worker nodes, you can use a placement configuration to control the fault domain and availability domain where they are created. Nodes will automatically come online with labels, which you can use to schedule your workloads so they are robust and highly available. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **Can I deploy private Kubernetes clusters?** Yes; with OKE, your Kubernetes clusters are integrated in your VCN. Your cluster worker nodes, load balancers, and the Kubernetes API endpoint are part of a private or public subnet of your VCN. Regular VCN routing and firewall rules control access to the Kubernetes API endpoint, making it accessible from a corporate network only, through a bastion host, or by specific platform services. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/

Claim 1	Accused Instrumentalities
	When should I use virtual nodes, managed nodes, or self-managed nodes? • **Virtual nodes** Virtual nodes offer a serverless Kubernetes experience. This option is ideal if you'd rather focus on your application and avoid managing the underlying infrastructure. Virtual nodes relieve you of management-related tasks such as scaling, upgrading, patching, troubleshooting, and provisioning worker nodes. • **Managed nodes** Managed nodes are a good choice for general purpose workloads. They offer an extensive list of customizable configuration options that have been tested by the OKE service. Unlike fully managed virtual nodes, you share the management of worker nodes with OCI. OKE simplifies the management process through features such as on-demand cycling to automate worker node updates, cluster self-healing upon failure detection, autoscaling, and more. • **Self-managed nodes** Self-managed nodes offer access to the underlying infrastructure, configuration options, and compute shapes that aren't currently available to managed nodes. This includes access to specialized infrastructure, such as RDMA-enabled bare metal cluster networks or confidential compute shapes. This advanced control makes self-managed nodes ideal for specialized use cases that aren't supported with managed nodes. Note that with self-managed nodes, you are fully responsible for managing the worker nodes—without the automated features provided by managed or virtual nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **What are the storage options for virtual nodes?** OKE virtual nodes do not yet have persistent storage capabilities. However, there are plans to introduce support for attaching persistent volumes backed by OCI Block Storage and OCI File Storage. If your Kubernetes application requires persistent storage, it's advisable to use OKE managed nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/

Claim 1	Accused Instrumentalities

Supported Images for Managed Nodes

Kubernetes Engine supports the provisioning of worker nodes (managed nodes only) using some, but not all, of the latest Oracle Linux images provided by Oracle Cloud Infrastructure.

You can use these Oracle Linux images when provisioning managed nodes:

- OKE Images
- Platform Images
- Custom Images

https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengimagesshapes.htm

What is Docker?

A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly.

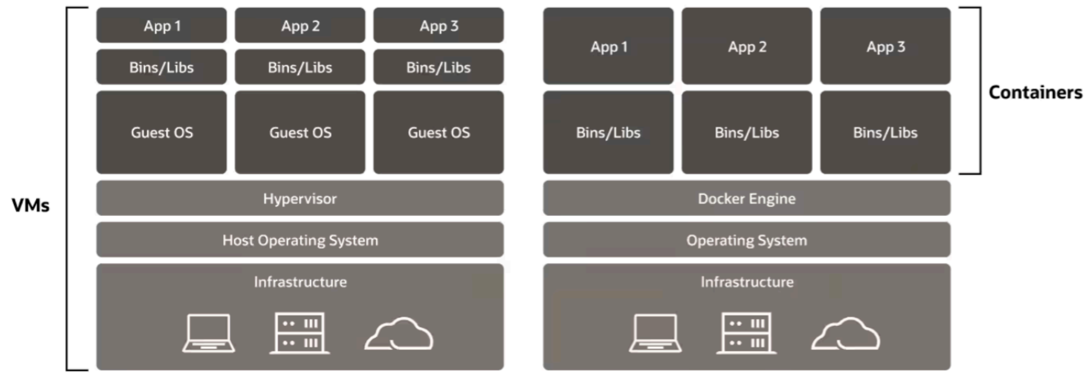
Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle.

Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products.

https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container: Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ **Docker versus Kubernetes** Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Case 5:24-cv-00739-P Document 63-4 Filed 02/24/25 Page 9 of 55

Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  **Virtual Machines** • Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system** **Containers** • Containers include the app and all of its dependencies, but **share the kernel** with other containers. • Run as an isolated process in userspace on the host operating system. • Not tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container Cloud Services The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container. To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming. To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services. Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode. https://www.techtarget.com/searchdatacenter/definition/kernel

Claim 1	Accused Instrumentalities
[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;	The method practiced by Oracle and/or its customer through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. For example, OCI and/or OKE stores application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent storage available to each node running the application. The terms “node” and “host” are both used to refer to the claimed server. The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network. In addition to the application software, each container includes associated system files, including a Linux user space required to execute the application, for example libc/glibc and other shared libraries, configuration files, etc. necessary for the application. For example, the container includes a base OS image provided by Oracle or by a third party. The container is compatible with the host kernel, for example because the container libraries are linked against the Linux kernel, and the supported host operating systems also use the Linux kernel, which has a stable binary interface. The containers are secure containers as claimed. For example, the data within an individual container is insulated from the effects of other containers except to the extent the container is specifically configured to allow other containers to modify its data, for example using a shared volume. *See, e.g.*: **Overview of File Storage** Oracle Cloud Infrastructure File Storage service provides a durable, scalable, secure, enterprise-grade network file system. You can connect to a File Storage service file system from any bare metal, virtual machine, or container instance in your Virtual Cloud Network (VCN). You can also access a file system from outside the VCN using VCN peering, Oracle Cloud Infrastructure FastConnect, and Internet Protocol security (IPSec) virtual private network (VPN). https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	What is Docker? A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly. Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle. Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

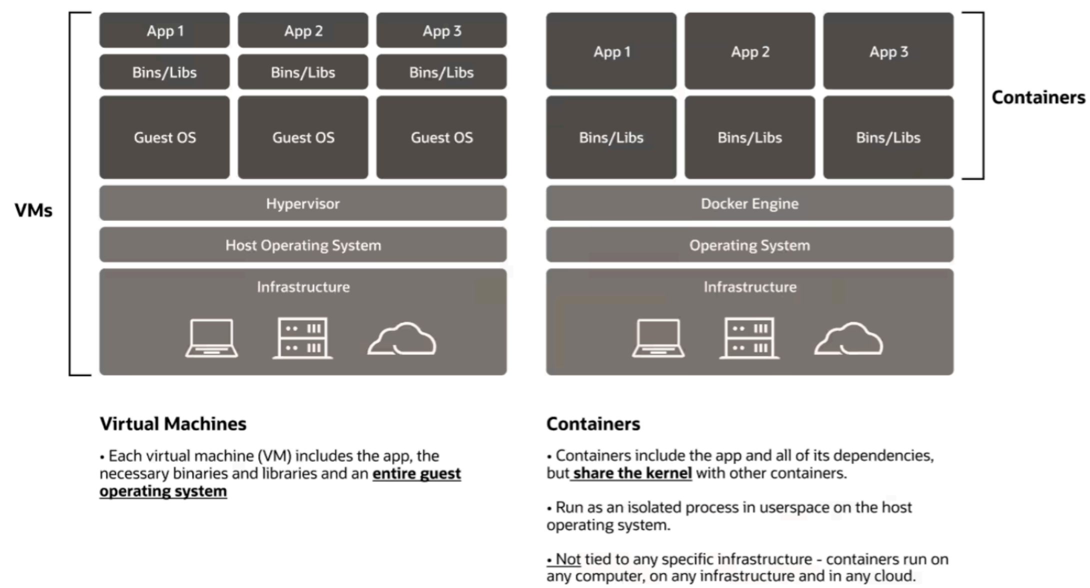
Claim 1	Accused Instrumentalities
	Container: Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ **Docker versus Kubernetes** Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>

Docker Basics

The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.

A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.



https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

</td></tr>
</table>

Claim 1	Accused Instrumentalities
	Docker **Images and Containers** An image is a read-only template with instructions for creating a Docker container and an image is based on another image. A container is a standard unit of software that packages up code and all its dependencies. Hence, the application runs quickly and reliably from one environment to another. A Docker Container Image is a lightweight, standalone, executable package of software that includes everything needed to run an application such as code, runtime, system tools, system libraries, and settings. Container images become containers at runtime and for Docker containers, the images become containers when they run on the engine. Containers are available for both Linux and Windows-based applications. The containerized software always runs the same code, regardless of the infrastructure. The container isolates software from its environment and ensures that it works uniformly despite differences for instance between Development, Staging, and Production. **Kubernetes (K8)** Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. It groups the containers that makes an application into logical units for easy management and discovery. https://docs.oracle.com/en/industries/financial-services/microservices-common/14.6.1.0.0/contg/technologies.html Container images A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings. https://kubernetes.io/docs/concepts/containers/

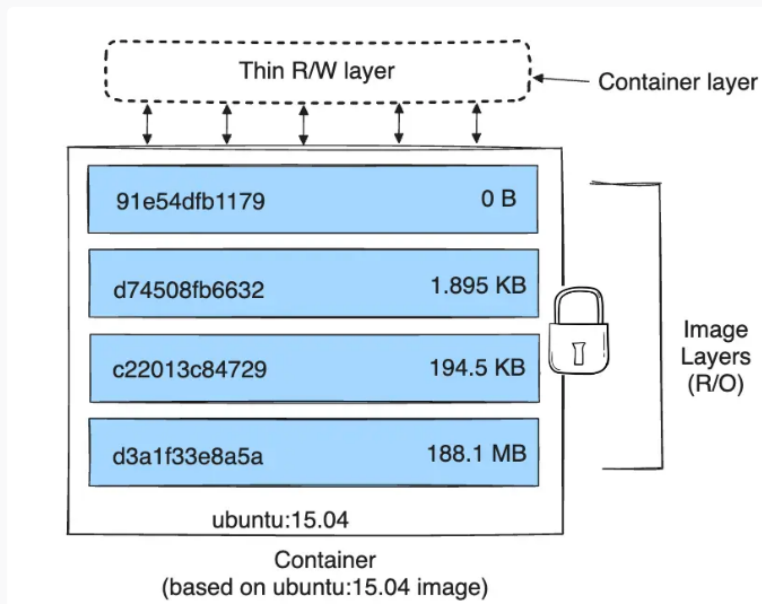
Case 4:24-cv-04740-PCP Document 83-4 Filed 12/20/25 Page 16 of 55

Claim 1	Accused Instrumentalities
	6. **Do Docker containers package up the entire OS and make it easier to deploy?** Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers. https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/ # About storage drivers To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way. ## Storage drivers versus Docker volumes Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer. Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance. https://docs.docker.com/storage/storagedriver/

Case 4:24-cv-04403 Document 1-4 Filed 12/10/25 Page 17 of 55

Claim 1	Accused Instrumentalities

	Images and layers
	
	A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:
	
	`# syntax=docker/dockerfile:1`
	
	`FROM ubuntu:22.04`
	`LABEL org.opencontainers.image.authors="org@example.com"`
	`COPY . /app`
	`RUN make /app`
	`RUN rm -r $HOME/.cache`
	`CMD python /app/app.py`
	
	This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.
	
	https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles]() and [use multi-stage builds]() sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/

Case 4:24-cv-07403-PCP Document 1-3-4 Filed 12/22/25 Page 19 of 56

Claim 1	Accused Instrumentalities
	# Volumes Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts: https://kubernetes.io/docs/concepts/storage/volumes/ ## Container environment The Kubernetes Container environment provides several important resources to Containers: • A filesystem, which is a combination of an image and one or more volumes. • Information about the Container itself. • Information about other objects in the cluster. https://kubernetes.io/docs/concepts/containers/container-environment/

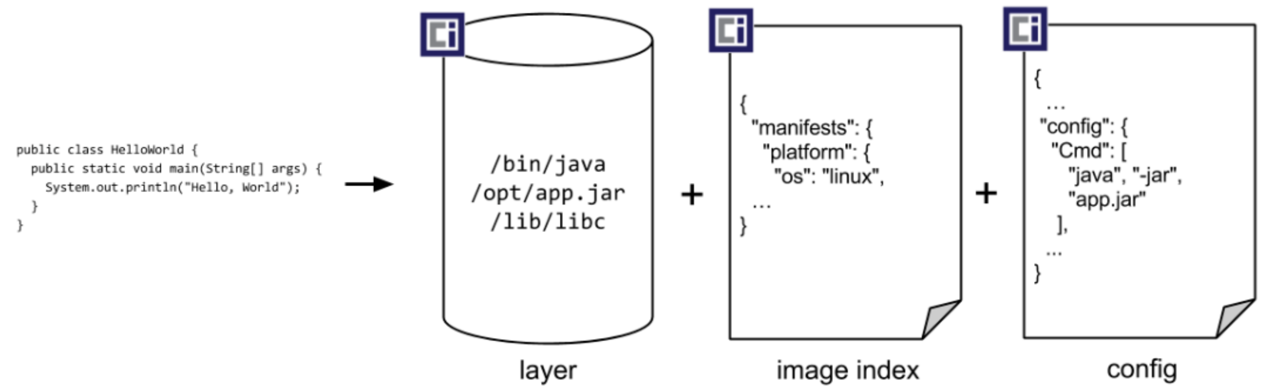
Claim 1	Accused Instrumentalities
	Images A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment. You typically create a container image of your application and push it to a registry before referring to it in a Pod. https://kubernetes.io/docs/concepts/containers/images/ **Volumes** On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a `Pod` and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested. https://kubernetes.io/docs/concepts/storage/volumes/

Claim 1	Accused Instrumentalities
	Open Container Initiative **Image Format Specification** This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration. The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>

Overview

At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.



https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md
</td></tr>
</table>

Claim 1	Accused Instrumentalities
	# OCI Image Configuration An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in [Layers](#). This section defines the `application/vnd.oci.image.config.v1+json` [media type](#). https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	Layer • Image filesystems are composed of *layers*. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. **Image JSON** • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	• **rootfs** *object*, REQUIRED The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash. ○ **type** *string*, REQUIRED MUST be set to `layers`. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image. ○ **diff_ids** *array of strings*, REQUIRED An array of layer content hashes (`DiffIDs`), in order from first to last. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

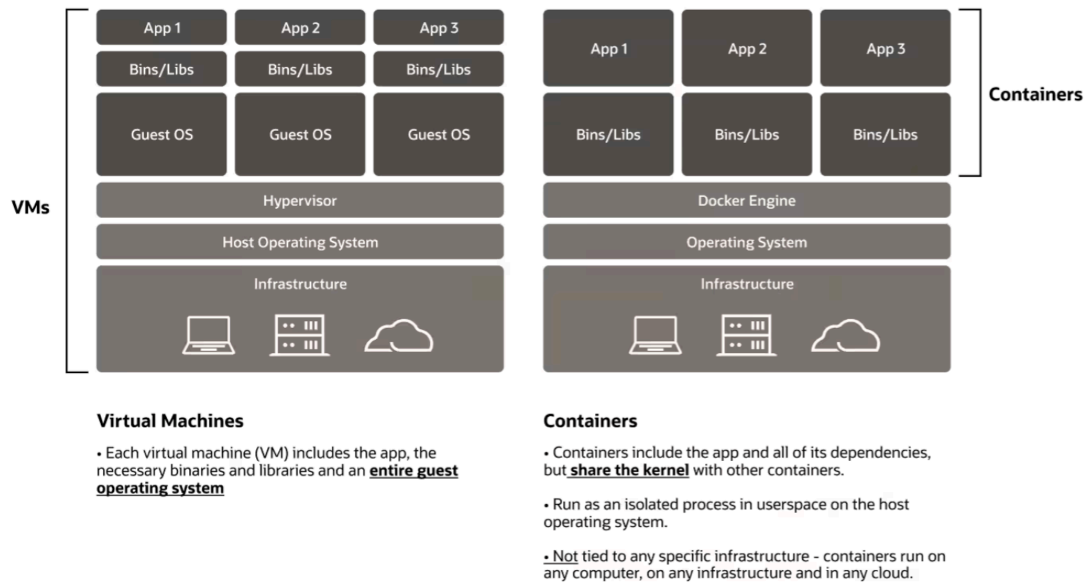
Claim 1	Accused Instrumentalities
[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The associated system files in the container are compatible with the host kernel, for example because they are linked against the Linux kernel and the supported host operating systems also use the Linux kernel, which has a stable binary interface. *See* discussion in element [1a] above. *See also, e.g.*: **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
[1c] the containers of application software excluding a kernel,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the containers of application software exclude a kernel. *See, e.g.*: **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ 6. **Do Docker containers package up the entire OS and make it easier to deploy?** Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers. https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/

Claim 1	Accused Instrumentalities
[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server. For example, each container will utilize its own associated system files, including libraries such as libc/glibc and configuration files, not the corresponding associated local system files (*e.g.*, libraries and configuration files of the host OS). As described above and below, in the Accused Instrumentalities the associated system files provide at least some of the same functionalities as the associated local system files. The host/node's associated local system files remain resident on the host/node, for example for use by system processes or applications outside the container environment. *See, e.g.*: **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Case 4:24-cv-07403-PCP Document 1-4 Filed 10/24/25 Page 30 of 56

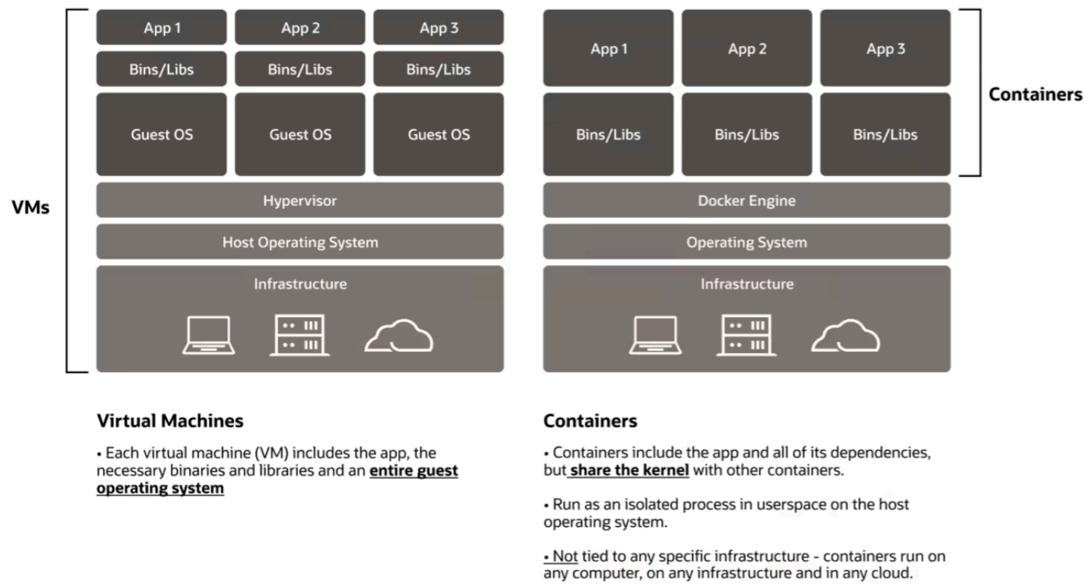
Claim 1	Accused Instrumentalities
	Docker versus Kubernetes Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container Cloud Services The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container. To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming. To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services. Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server. For example, in some cases the host OS and container will use one or more identical system files, for example when both the host and the container incorporate the same Linux distribution version, or when both host and container use the same version of libc. In the case where the associated system files are identical to the associated local system files, they are copies thereof. In other cases modified copies are used instead, for example when different versions of the same library, or configuration files with different parameters, are used by the host and container. *See, e.g.*: **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker versus Kubernetes Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

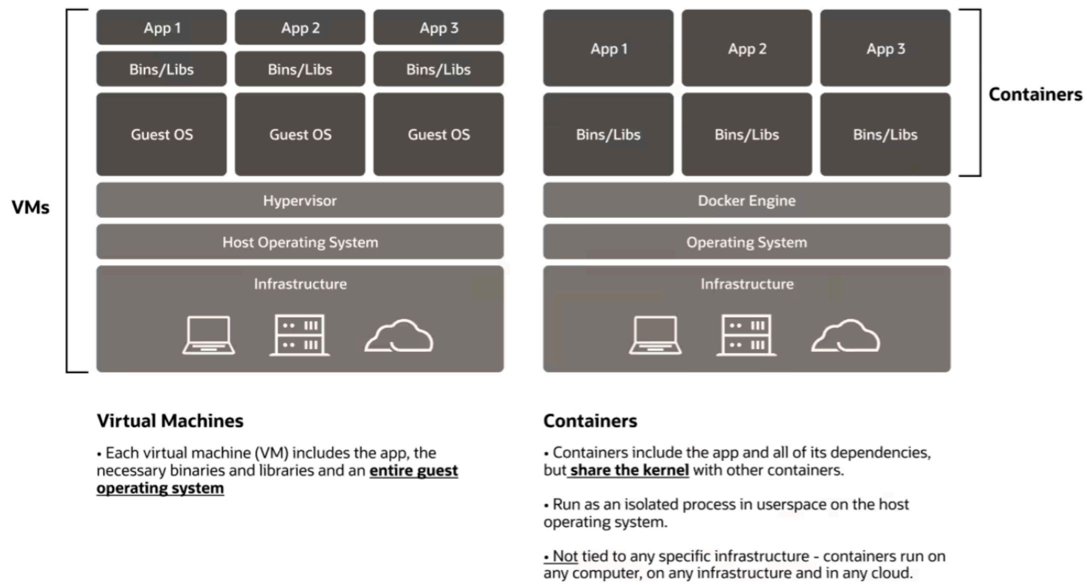
Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container Cloud Services The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container. To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming. To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services. Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ `COPY` and `ADD`: These commands copy files and directories from your local filesystem into the Docker image. They are often used to include your application code, configuration files, and dependencies. https://medium.com/@swalperen3008/what-is-dockerize-and-dockerize-your-project-a-step-by-step-guide-899c48a34df6

Claim 1	Accused Instrumentalities
	Container images A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings. https://kubernetes.io/docs/concepts/containers/

Claim 1	Accused Instrumentalities
[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software. For example, each container has an isolated runtime environment that cannot be accessed by other containers, for example including a per-container writeable layer or other ephemeral per-container storage. For another example, when the plurality of secure containers each corresponds to a different container image, each container cannot access another container's image and therefore application software. *See, e.g.*: **Cgroups and Namespaces History** The underlying Linux kernel features that Docker uses are cgroups and namespaces. In 2008 cgroups were introduced to the Linux kernel based on work previously done by Google developers[1]. Cgroups limit and account for the resource usage of a set of operating system processes. The Linux kernel uses namespace to isolate the system resources of processes from each other. The first namespace, i.e. the mount namespace, was introduced as early as 2002.[2] https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container: Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ **Docker versus Kubernetes** Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	About storage drivers To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way. **Storage drivers versus Docker volumes** Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer. Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance. https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities

Images and layers

A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:

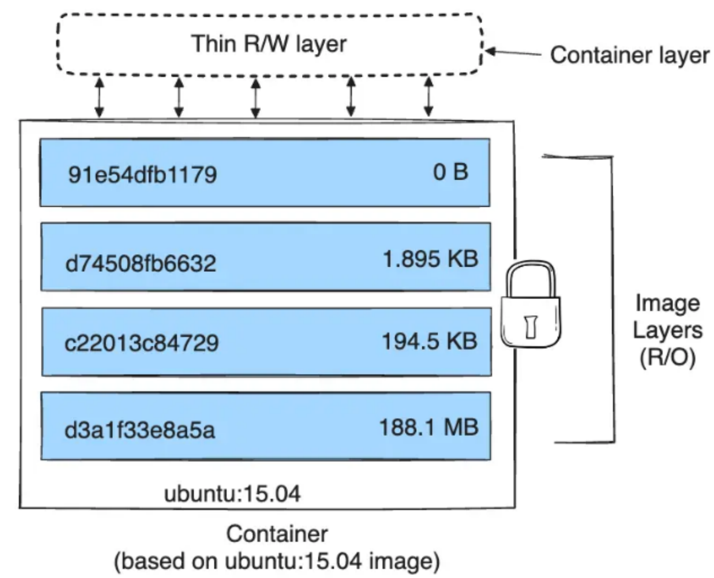
```
# syntax=docker/dockerfile:1

FROM ubuntu:22.04
LABEL org.opencontainers.image.authors="org@example.com"
COPY . /app
RUN make /app
RUN rm -r $HOME/.cache
CMD python /app/app.py
```

This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.

https://docs.docker.com/storage/storagedriver/

Case 4:24-cv-07003-PCP Document 1-3-4 Filed 12/20/24/25 Page 42 of 55

Claim 1	Accused Instrumentalities
	Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system. For example, the container's root file system comprises the image layer(s), an ephemeral writeable layer (e.g., in Docker terminology the container layer), and optionally one or more volumes. This root file system is distinct and isolated from the host operating system's root file system. *See, e.g.*:

<table>
<tr><th>Claim 1</th><th>Accused Instrumentalities</th></tr>
<tr><td></td><td>

Setting Up Storage for Kubernetes Clusters

Find out how to define and apply persistent volume claims to clusters you've created using Kubernetes Engine (OKE). With Oracle Cloud Infrastructure as the underlying IaaS provider, you can provision persistent volume claims by attaching volumes from the Block Volume service or by mounting file systems from the File Storage service.

Container storage via a container's root file system is ephemeral, and can disappear upon container deletion and creation. To provide a durable location to prevent data from being lost, you can create and use persistent volumes to store data outside of containers.

A persistent volume offers persistent storage that enables your data to remain intact, regardless of whether the containers to which the storage is connected are terminated.

A persistent volume claim (PVC) is a request for storage, which is met by binding the PVC to a persistent volume (PV). A PVC provides an abstraction layer to the underlying storage.

With Oracle Cloud Infrastructure, you can provision persistent volume claims:

- By attaching volumes from the Oracle Cloud Infrastructure Block Volume service. The volumes are connected to clusters created by Kubernetes Engine using CSI (Container Storage Interface) or FlexVolume volume plugins deployed on the clusters. Oracle recommends the CSI volume plugin since the upstream Kubernetes project deprecates the FlexVolume volume plugin in Kubernetes version 1.23. See Provisioning PVCs on the Block Volume Service.
- By mounting file systems in the Oracle Cloud Infrastructure File Storage service. The File Storage service file systems are mounted inside containers running on clusters created by Kubernetes Engine using a CSI (Container Storage Interface) volume plugin deployed on the clusters. See Provisioning PVCs on the File Storage Service.

https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengcreatingpersistentvolumeclaim.htm

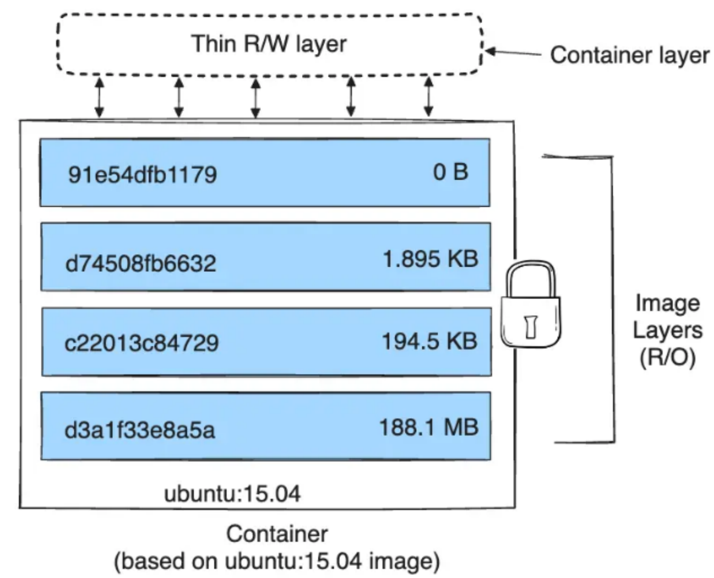
</td></tr>
</table>

Claim 1	Accused Instrumentalities
	## Configuring Docker Storage The Docker Engine is configured to use `overlay2` as the default storage driver to manage Docker containers. This provides a performance and scalability improvement on earlier releases that used the device mapper as the default storage driver, but the technology is new and should be tested properly before use in production environments. For more information on `overlay2`, see: https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/ Overlay file systems can corrupt when used in conjunction with any file system that does not have dtype support enabled. https://docs.oracle.com/en/operating-systems/oracle-linux/docker/docker-InstallingOracleContainerRuntimeforDocker.html In Kubernetes, each container can read and write to its own file system. But when a container is restarted, all data is lost. Therefore, containers that need to maintain state would store data in a persistent storage such as Network File System (NFS). What's already stored in NFS isn't deleted when a pod, which might contain one or more containers, is destroyed. Also, an NFS can be accessed from multiple pods at the same time, so an NFS can be used to share data between pods. This behavior is really useful when containers or applications need to read configuration data from a single shared file system or when multiple containers need to read from and write data to a single shared file system. Oracle Cloud Infrastructure File Storage provides a durable, scalable, and distributed enterprise-grade network file system that supports NFS version 3 along with Network Lock Manager (NLM) for a locking mechanism. You can connect to File Storage from any bare metal, virtual machine, or container instance in your virtual cloud network (VCN). You can also access a file system from outside the VCN by using Oracle Cloud Infrastructure FastConnect or an Internet Protocol Security (IPSec) virtual private network (VPN). File Storage is a fully managed service so you don't have to worry about hardware installations and maintenance, capacity planning, software upgrades, security patches, and so on. You can start with a file system that contains only a few kilobytes of data and grows to handle 8 exabytes of data. https://blogs.oracle.com/cloud-infrastructure/post/using-file-storage-service-with-container-engine-for-kubernetes

Claim 1	Accused Instrumentalities
	About storage drivers To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way. **Storage drivers versus Docker volumes** Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer. Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance. https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities

	# Images and layers
	
	A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:
	
	`# syntax=docker/dockerfile:1`
	
	`FROM ubuntu:22.04`
	`LABEL org.opencontainers.image.authors="org@example.com"`
	`COPY . /app`
	`RUN make /app`
	`RUN rm -r $HOME/.cache`
	`CMD python /app/app.py`
	
	This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.
	
	https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles] and [use multi-stage builds] sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/

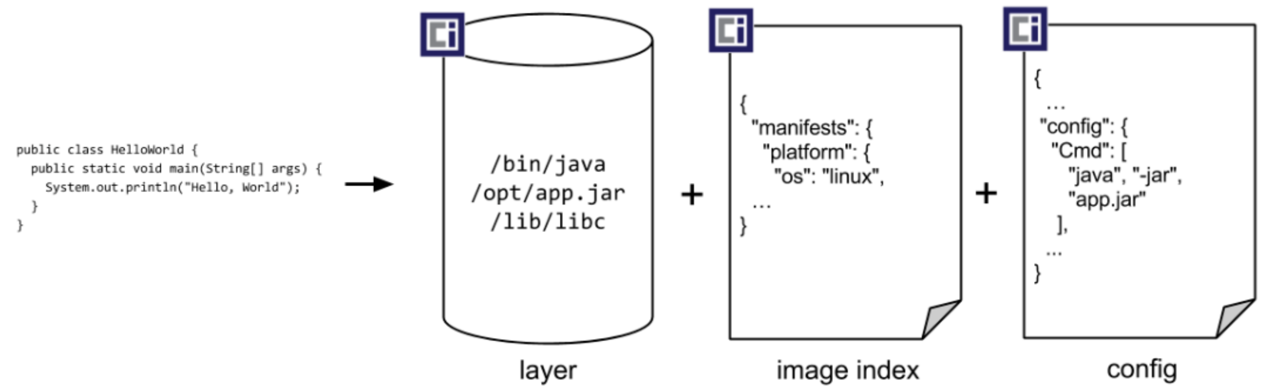
Claim 1	Accused Instrumentalities
	Volumes Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts: https://kubernetes.io/docs/concepts/storage/volumes/ **Container environment** The Kubernetes Container environment provides several important resources to Containers: • A filesystem, which is a combination of an image and one or more volumes. • Information about the Container itself. • Information about other objects in the cluster. https://kubernetes.io/docs/concepts/containers/container-environment/

Case 4:24-cv-07003-JST Document 23-4 Filed 12/20/24 Page 51 of 56

Claim 1	Accused Instrumentalities
	Images A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment. You typically create a container image of your application and push it to a registry before referring to it in a Pod. https://kubernetes.io/docs/concepts/containers/images/ **Volumes** On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a `Pod` and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested. https://kubernetes.io/docs/concepts/storage/volumes/

Claim 1	Accused Instrumentalities
	# Open Container Initiative ## Image Format Specification This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration. The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Case 4:24-cv-07403 Document 1-4 Filed 10/24/25 Page 52 of 55

Claim 1	Accused Instrumentalities
	## Overview At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.  https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	# OCI Image Configuration An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers. This section defines the `application/vnd.oci.image.config.v1+json` media type. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	Layer • Image filesystems are composed of *layers*. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. **Image JSON** • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	• **rootfs** *object*, REQUIRED The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash. ○ **type** *string*, REQUIRED MUST be set to `layers`. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image. ○ **diff_ids** *array of strings*, REQUIRED An array of layer content hashes (`DiffIDs`), in order from first to last. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md